2015年计算机组成研讨班

多周期CPU形式建模综合方法

多周期数据通路、RTL、时序分析

高小鹏
gxp@buaa.edu.cn

北京航空航天大学计算机学院
2015年7月

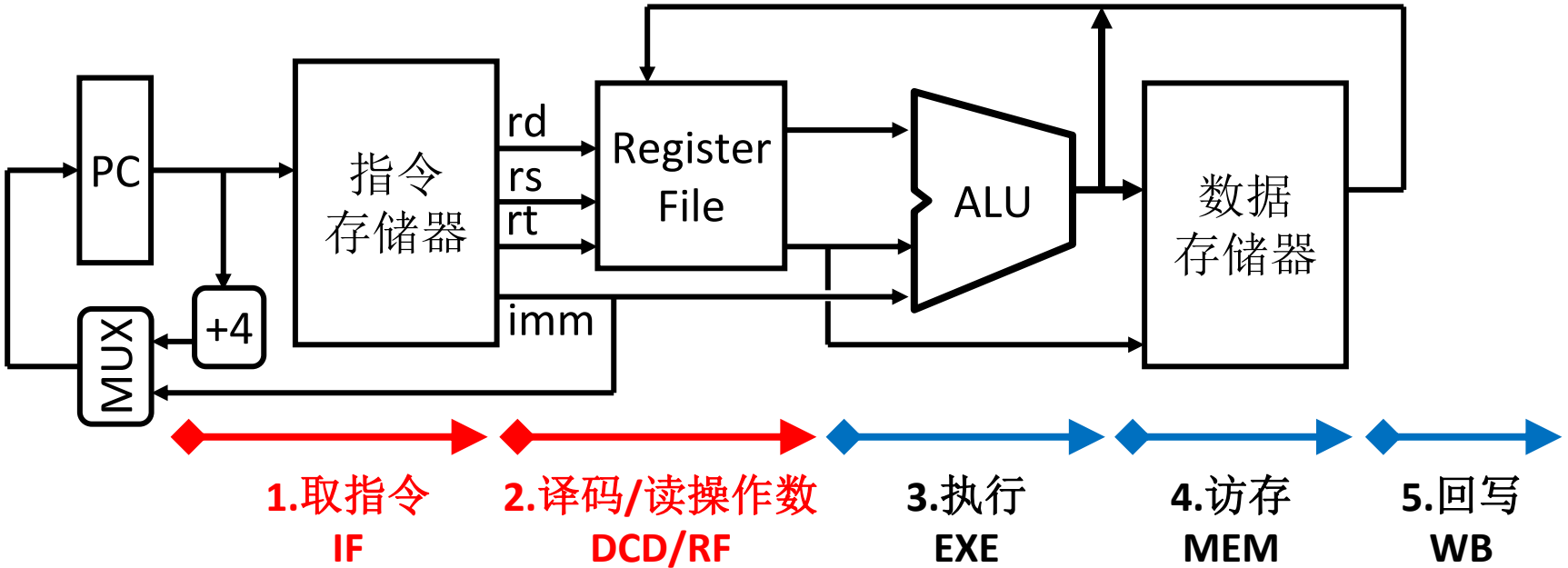
目录

- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

回顾：单周期数据通路

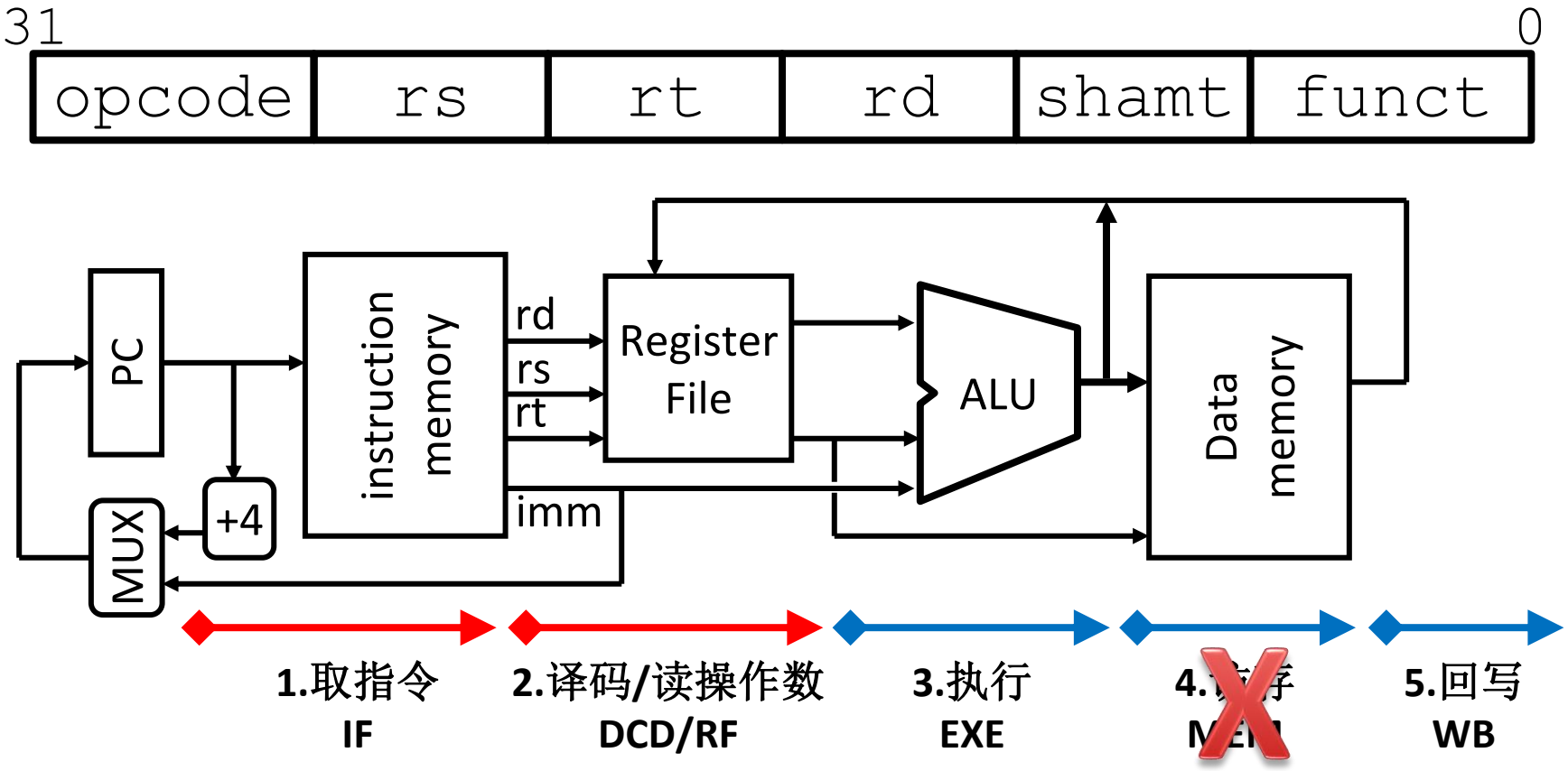
- 从逻辑上：数据通路总共为5个逻辑步骤
- 公共步骤：取指令、译码/读操作数
  - 所有指令均必须经历的2个步骤

取指令	IF	PC驱动IM读取指令
译码/读操作数	DCD/RF	译码属于控制器范畴；可以与读操作数并行
执行	EXE	ALU完成算数/逻辑运算
访存	MEM	读DM或写DM
回写	WB	ALU计算结果或IM读出数据写入寄存器堆



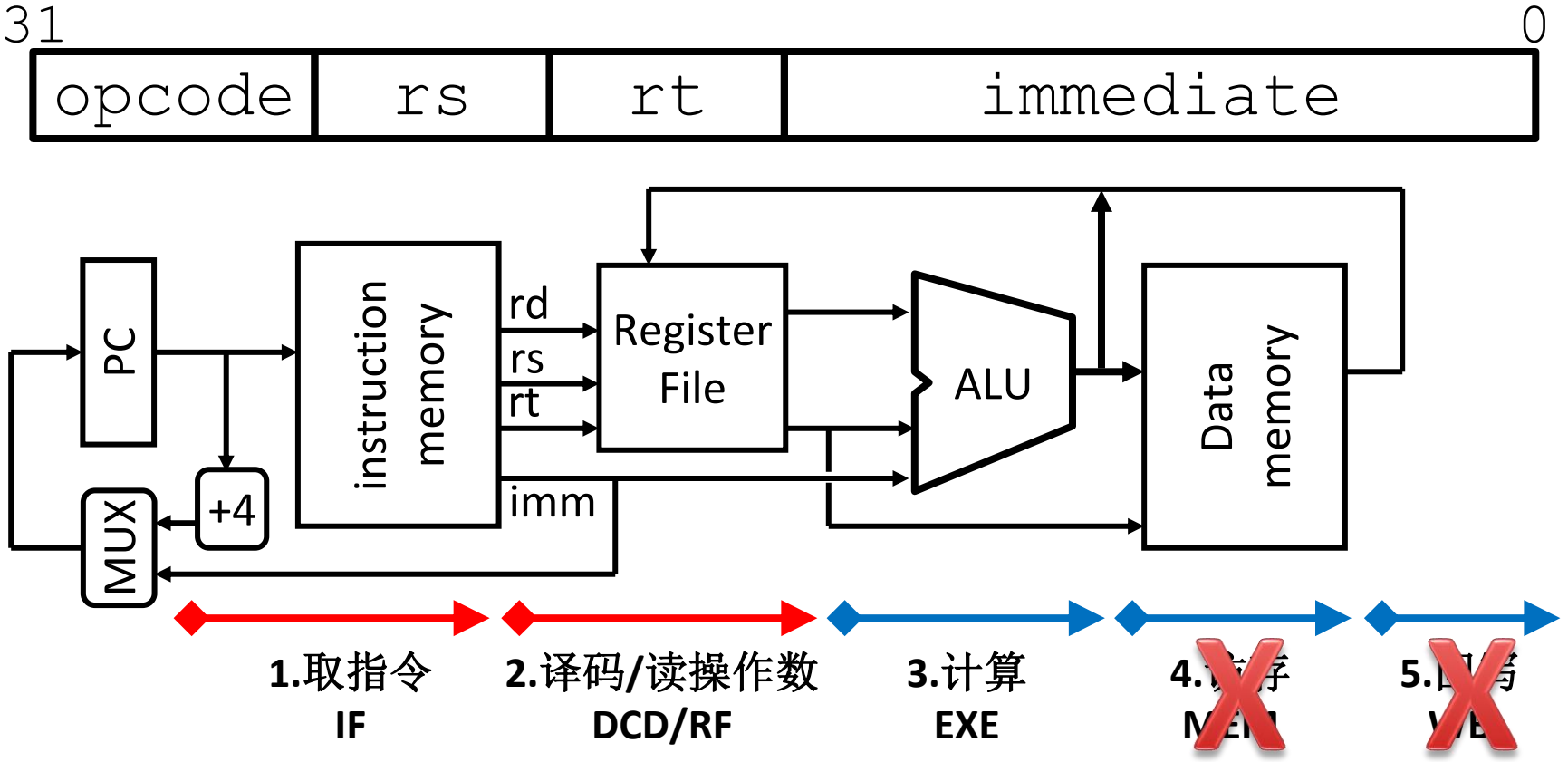
运算类指令：理想执行过程

- 指令：ADD、SUB、OR。。。
- 需求：R[rd] ← R[rs] op R[rt]
- 过程：取指、译码/读寄存器、执行、~~访存~~、回写



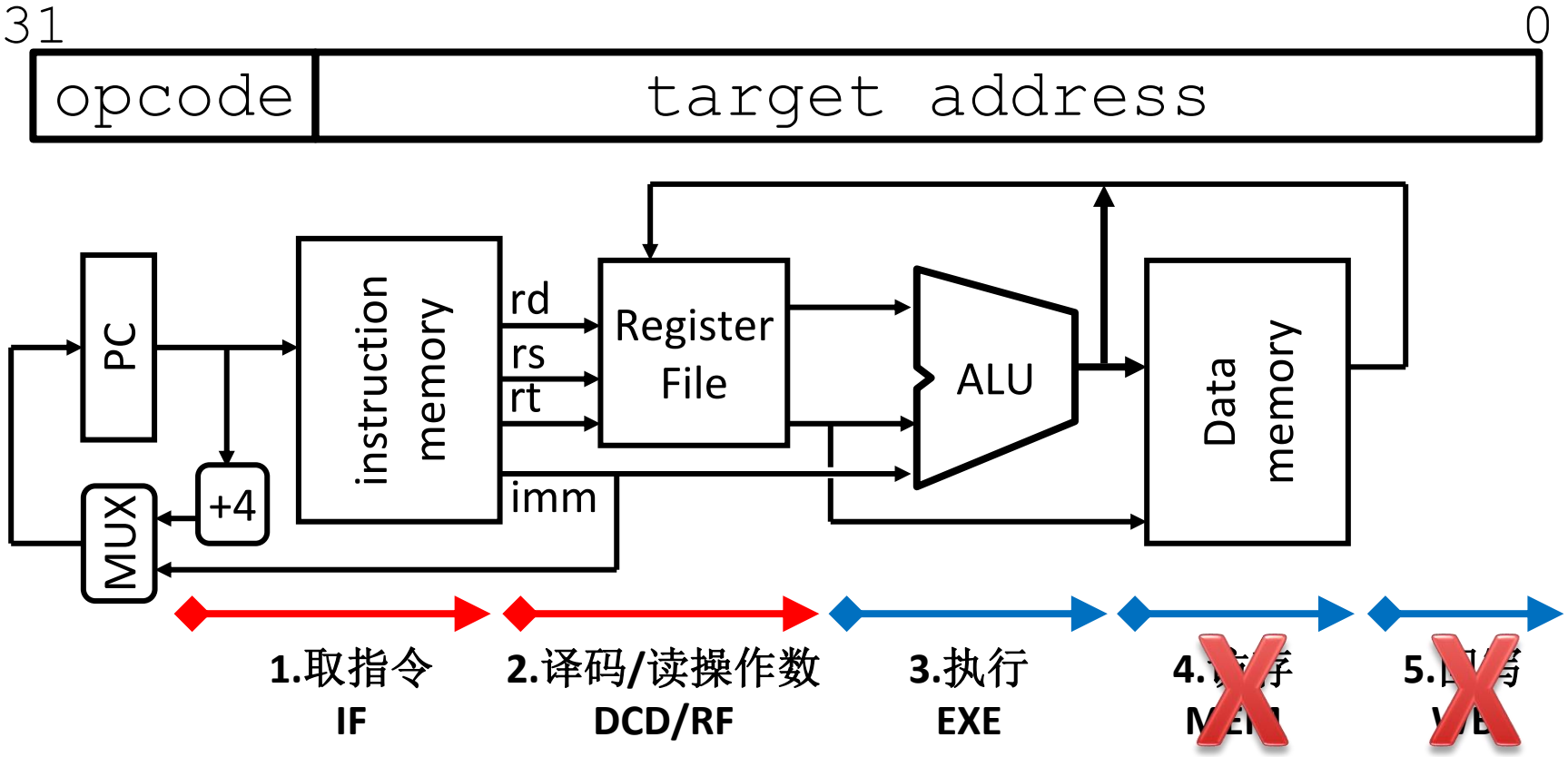
分支类指令：理想执行过程

- 指令：BEQ、。。。
- 需求：PC ← 条件? PC + Ext(Imm) : PC + 4
- 过程：取指、译码/读寄存器、执行、~~访存、回写~~



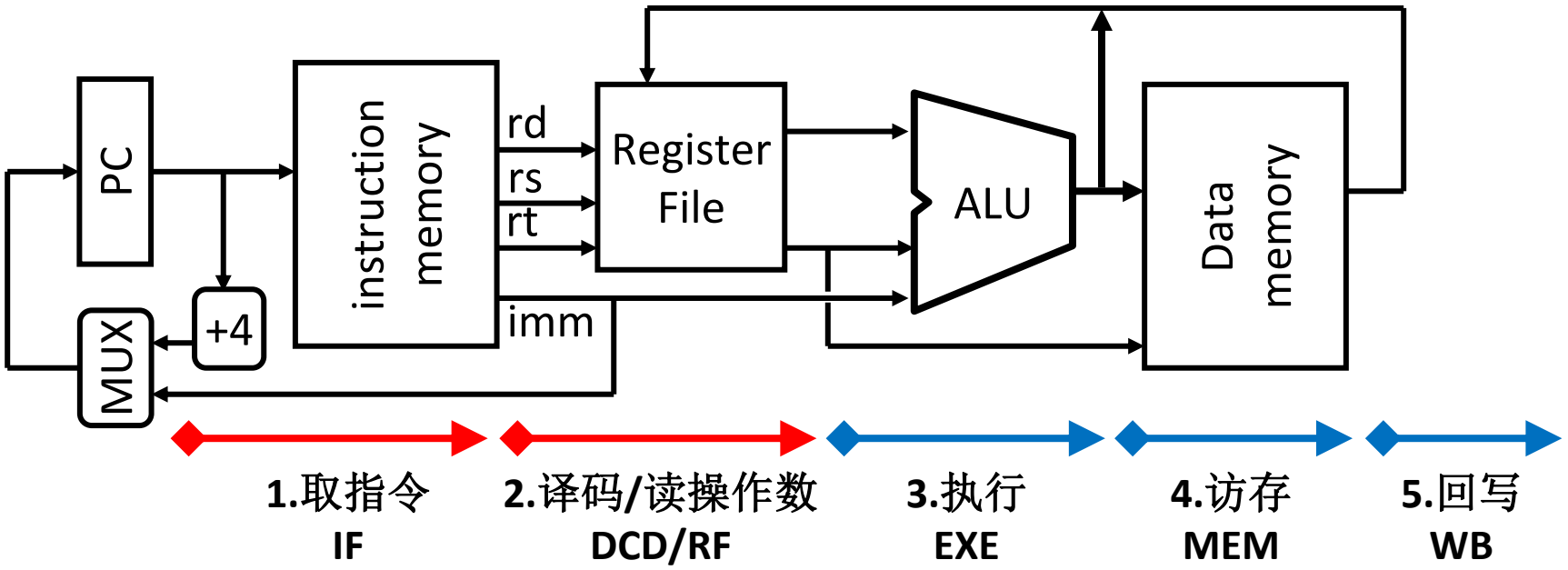
跳转指令：理想执行过程

- 指令：J
- 需求：PC ← PC[31:28] || target_address || 00
- 过程：取指、译码/读寄存器、执行、~~访存~~、~~回写~~



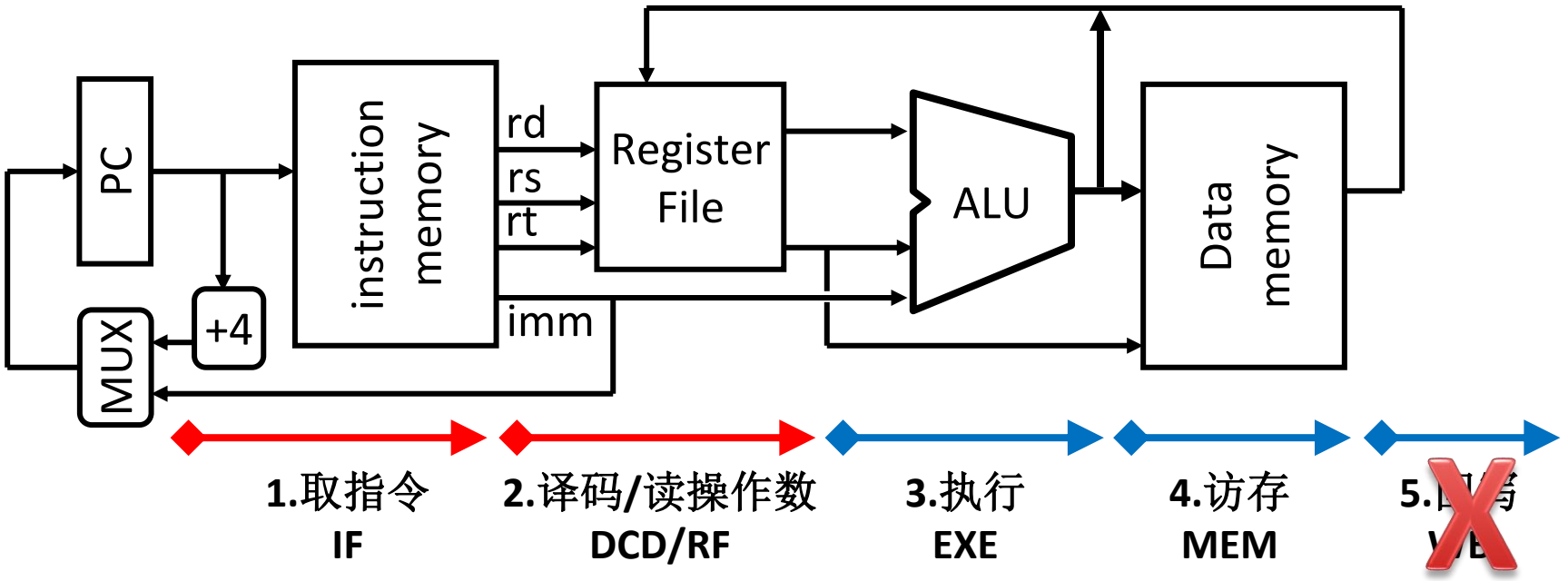
读存储指令：理想执行过程

- 指令：LW
- 需求：RF[rt] ← memory[RF[base] + offset]
- 过程：取指、译码/读寄存器、执行、访存、回写



写存储指令：理想执行过程

- 指令：SW
- 需求：memory[RF[base] + offset] ← RF[rt]
- 过程：取指、译码/读寄存器、执行、访存、~~回写~~



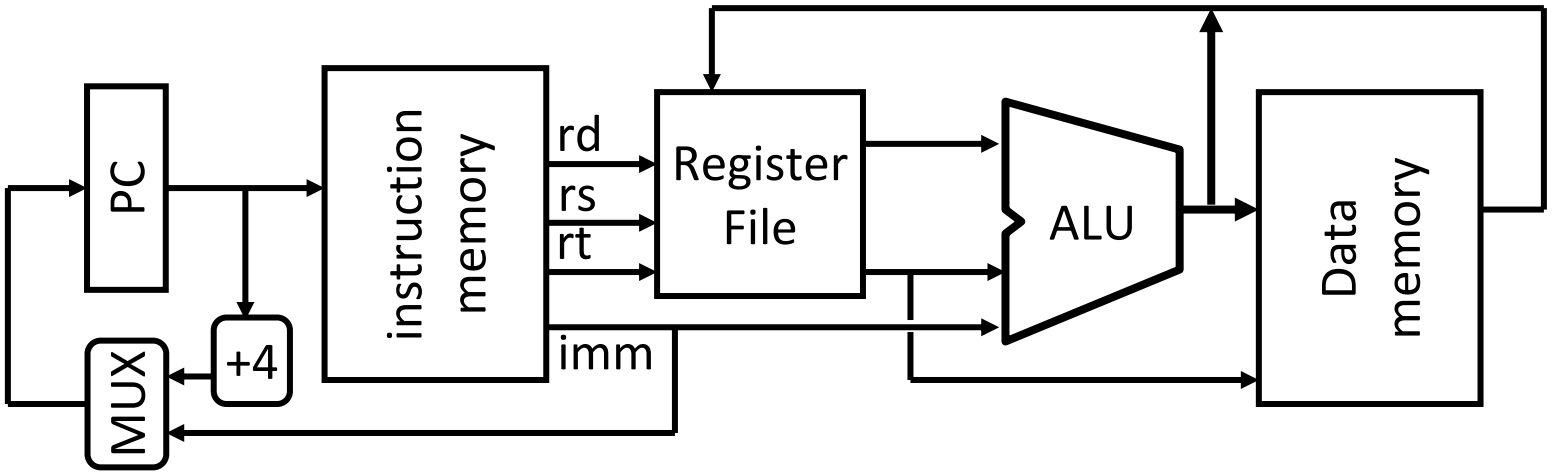
物理执行路径 vs. 理想执行过程

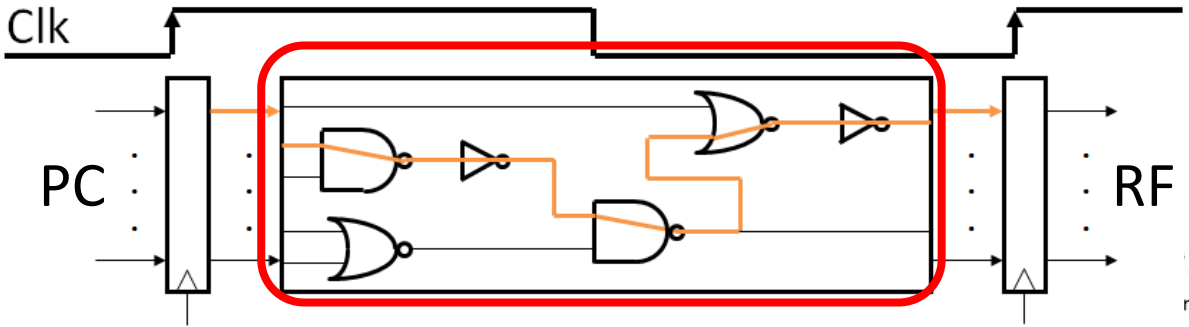
- 2个现象
  - 现象1：不同指令的理想执行过程不同
  - 现象2：所有指令都有前3个阶段
    - 前2阶段完全相同；第3阶段功能有差异
- 启示
  - 不同指令的理想执行过程不同，即理想执行时间不同
  - 不同指令能否具有不同物理执行路径，以对应理想执行过程？

	IF	DCD/RF	EXE	MEM	WB
计算	✓	✓	✓		✓
分支	✓	✓	✓		
跳转	✓	✓	✓		?
读存储	✓	✓	✓	✓	✓
写存储	✓	✓	✓	✓	

单周期数据通路的缺陷

- 模型：PC → 组合逻辑 → 寄存器堆
  - 单一组合逻辑实现了全部5个阶段的逻辑功能
  - 必然存在关键路径，且关键路径导致每条指令延迟均相同
- 结论：无法利用不同指令具有不同执行需求的潜在特性



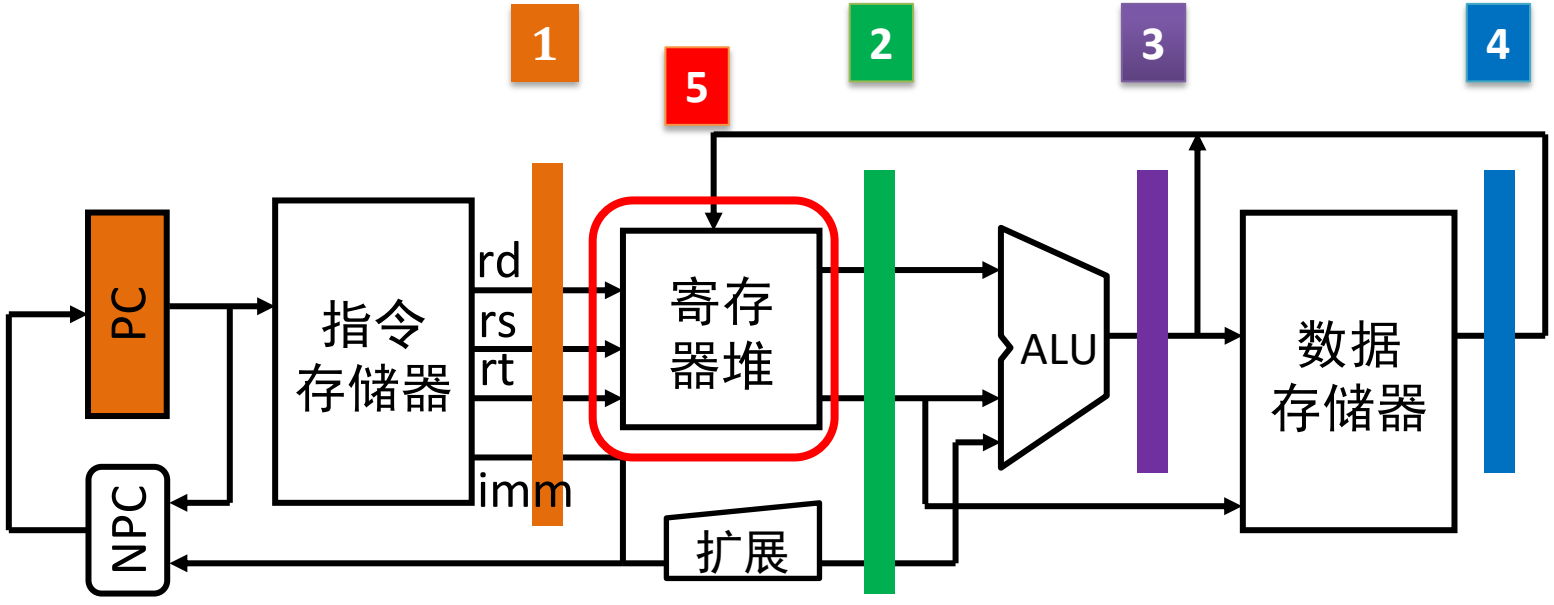


目录

- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

多周期数据通路构思

- 单周期的单一路径被物理切分为多段路径
  - 在数据通路上插入多个寄存器
  - 单一组合逻辑被切分为多段组合逻辑
  - 单一关键路径的大延迟变为多个分段路径的小延迟



TIP

RF：在读寄存器阶段，表现为组合逻辑；在回写阶段，表现为寄存器

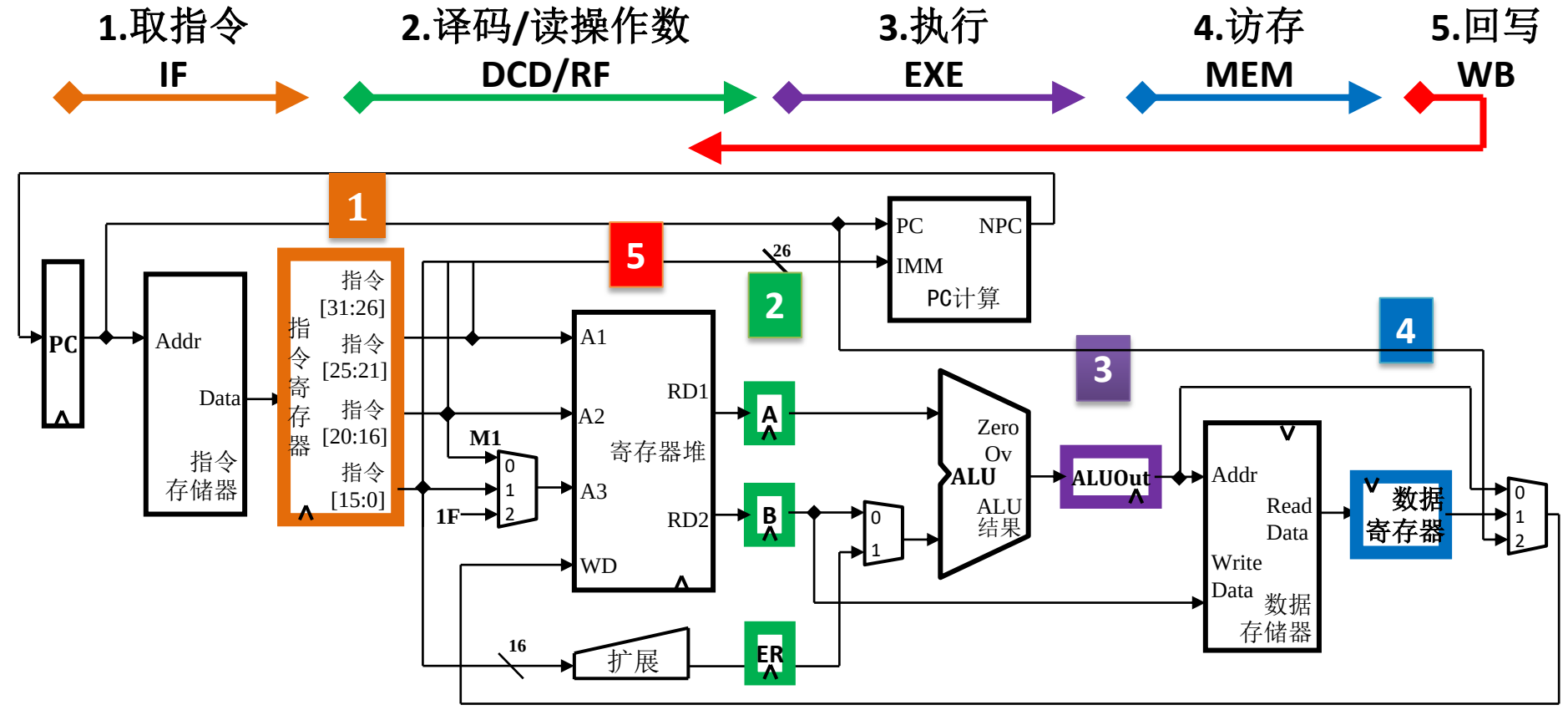
需要的寄存器

- 每个功能部件后面都插入寄存器
  - RF和DM：虽然是时序部件，但在读出操作时表现为组合逻辑

组合逻辑	插入的寄存器	用途
IM	IR	保存指令
RF（读）	A和B	保存2个寄存器值
扩展单元	ER	保存32位扩展值
ALU	ALUOut	保存计算结果
DM（读）	DR	保存读出的数据

基础多周期数据通路

- NPC（PC计算）：完成与PC相关的一切计算
  - PC+4；PC+imm16；PC+imm26



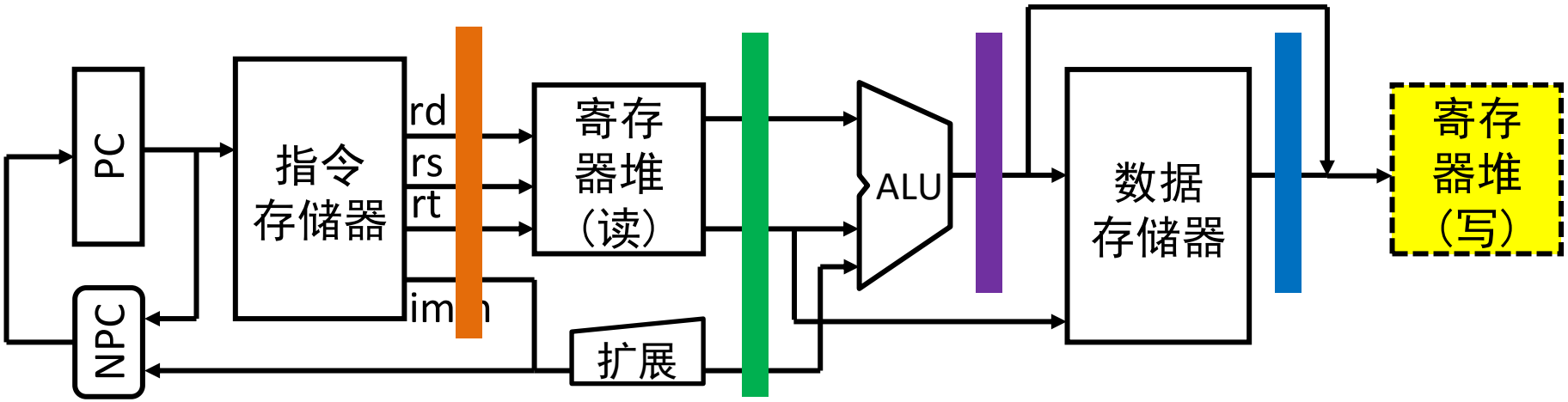
多周期数据通路特点

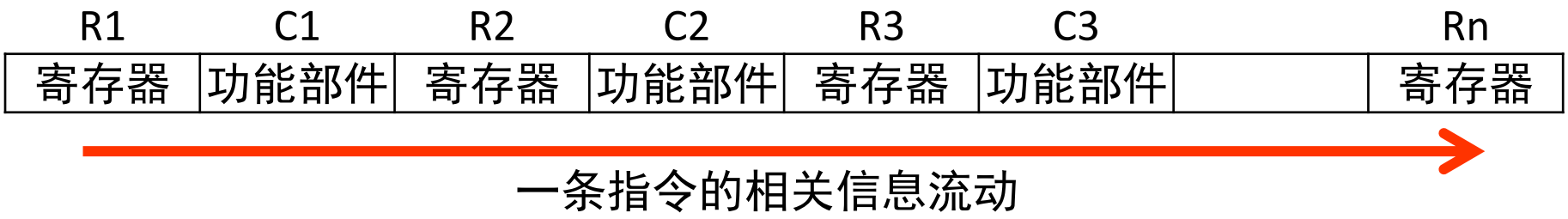
- 段内执行： 每段1个Cycle，段内模型R→C→R
  - 寄存器输出→组合逻辑→寄存器写入
  - 组合逻辑：如NPC、ALU等
    - RF/DM等：在读操作时，其行为为组合逻辑
- 段间执行：存在逻辑依赖关系
  - 前段没有执行，后段执行无意义
    - 不是不能执行，而是执行结果无意义
    - 例如：指令不读入IR，读操作数就无意义
- 根据指令需求组合分段，构成相应通路
  - 不同指令执行，占用不同的功能单元
  - 很好的映射不同指令的理想执行过程
  - 共有阶段：阶段1(读取指令)、阶段2(读操作数)

TIP
逻辑依赖是分析数据通路构造的关键。

多周期数据通路特点

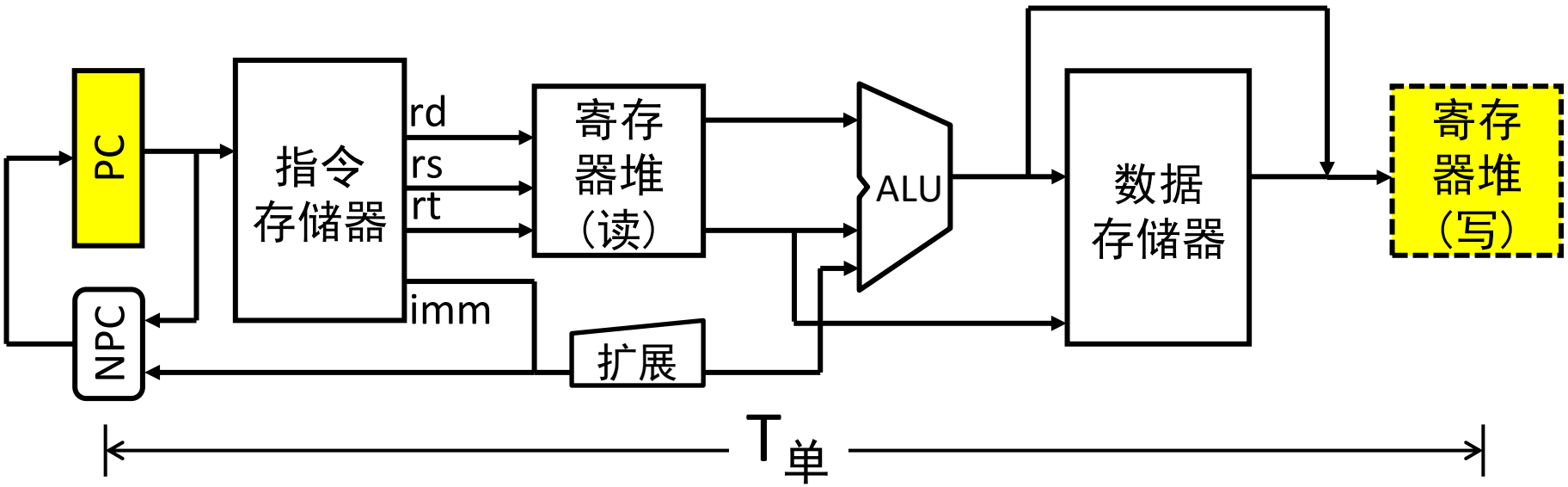
- 在推导某条指令的具体执行路径时，对于任意一级寄存器R_i，只要前面任意一级的寄存器/组合逻辑包含有需要的信息，均可以将相关信息直接输入到





多周期数据通路的优势

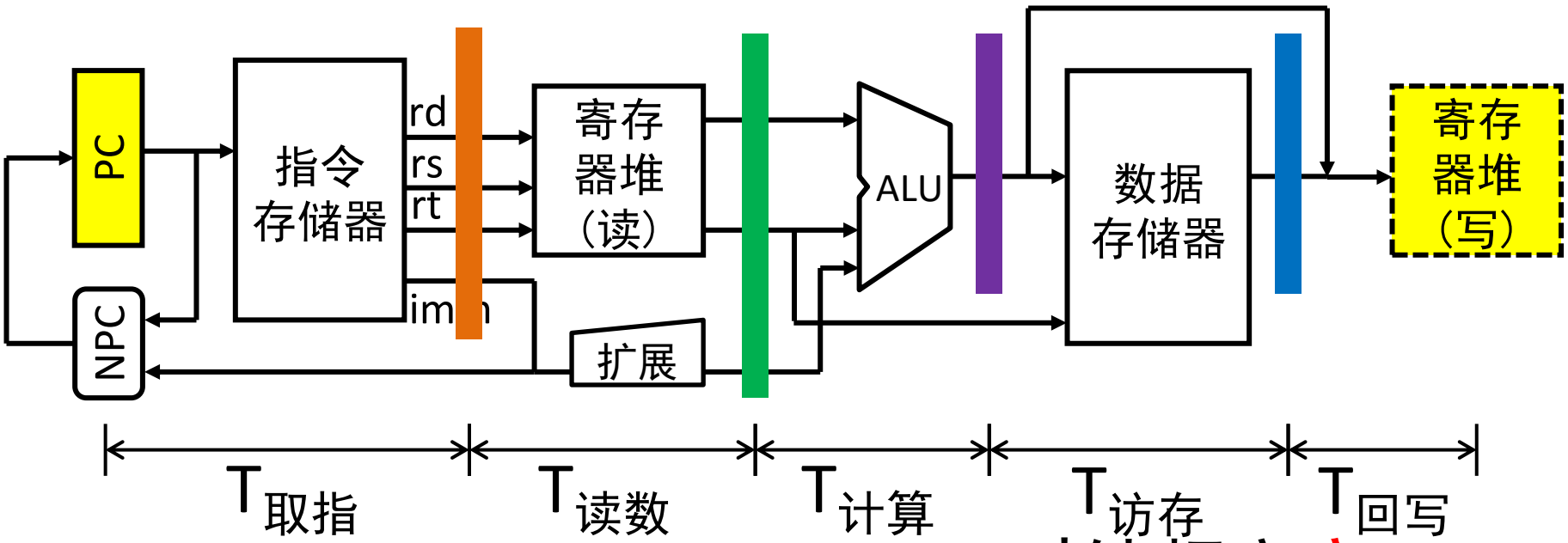
- $T_{单}$ = 关键路径=T_{lw}



- $f_{单}$ = $1/T_{单}$
- 虽然每条指令都只需1个cycle，但时钟频率低
  - CPI = 1

多周期数据通路的优势

- $T_{多} = MAX\{T_{取指}, T_{读数}, T_{计算}, T_{访存}, T_{回写}\}$



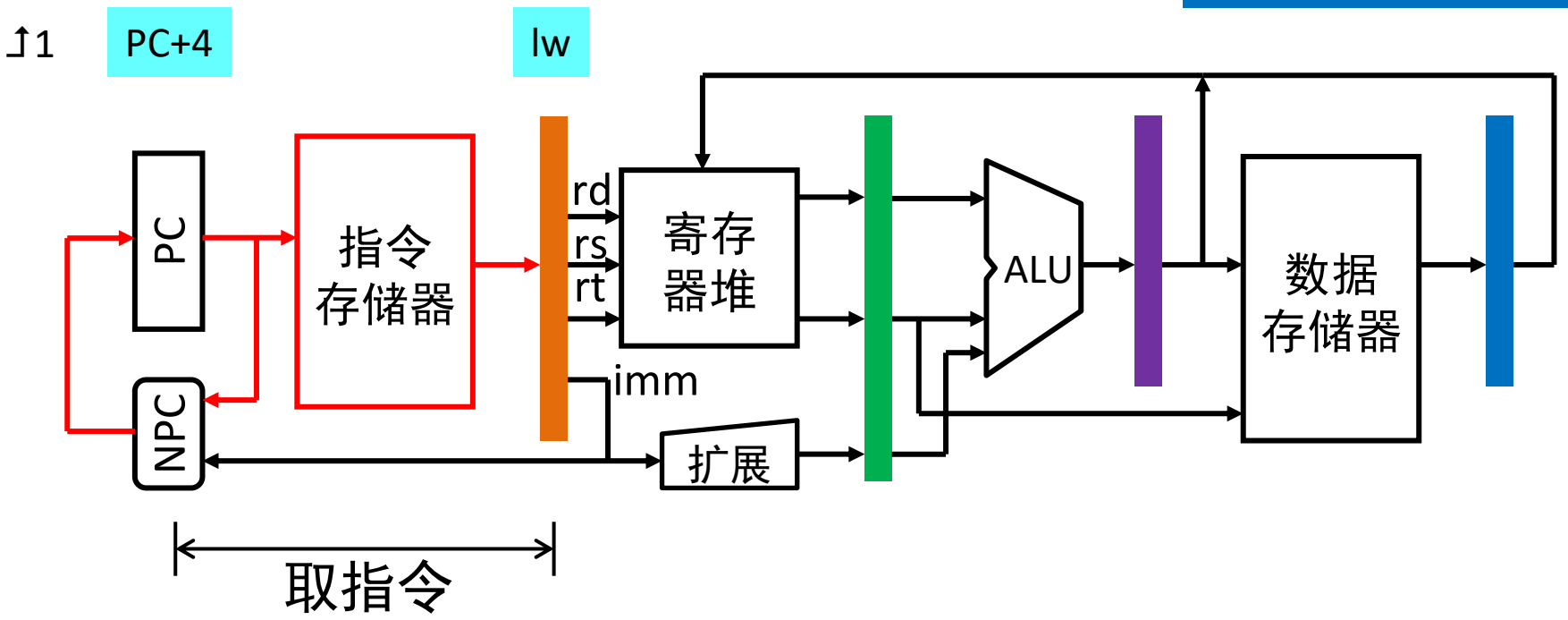
- $T_{多} \approx T_{单}/5$， $f_{多} = 1/T_{多} \approx 5f_{单}$；时钟频率高
- 每个指令需要多个cycle
  - CPI > 1
  - lw：最多，5个cycle，CPI=5

TIP
以add为例，ALU结果可以直接在下个cycle回写至RF。因此在多周期数据通路中，每条指令的CPI可以不同。

指令执行案例分析：lw

- Cycle 1：取指令
  - PC驱动IM，读出lw，并写入IR
  - PC驱动NPC，计算PC+4，并写入PC

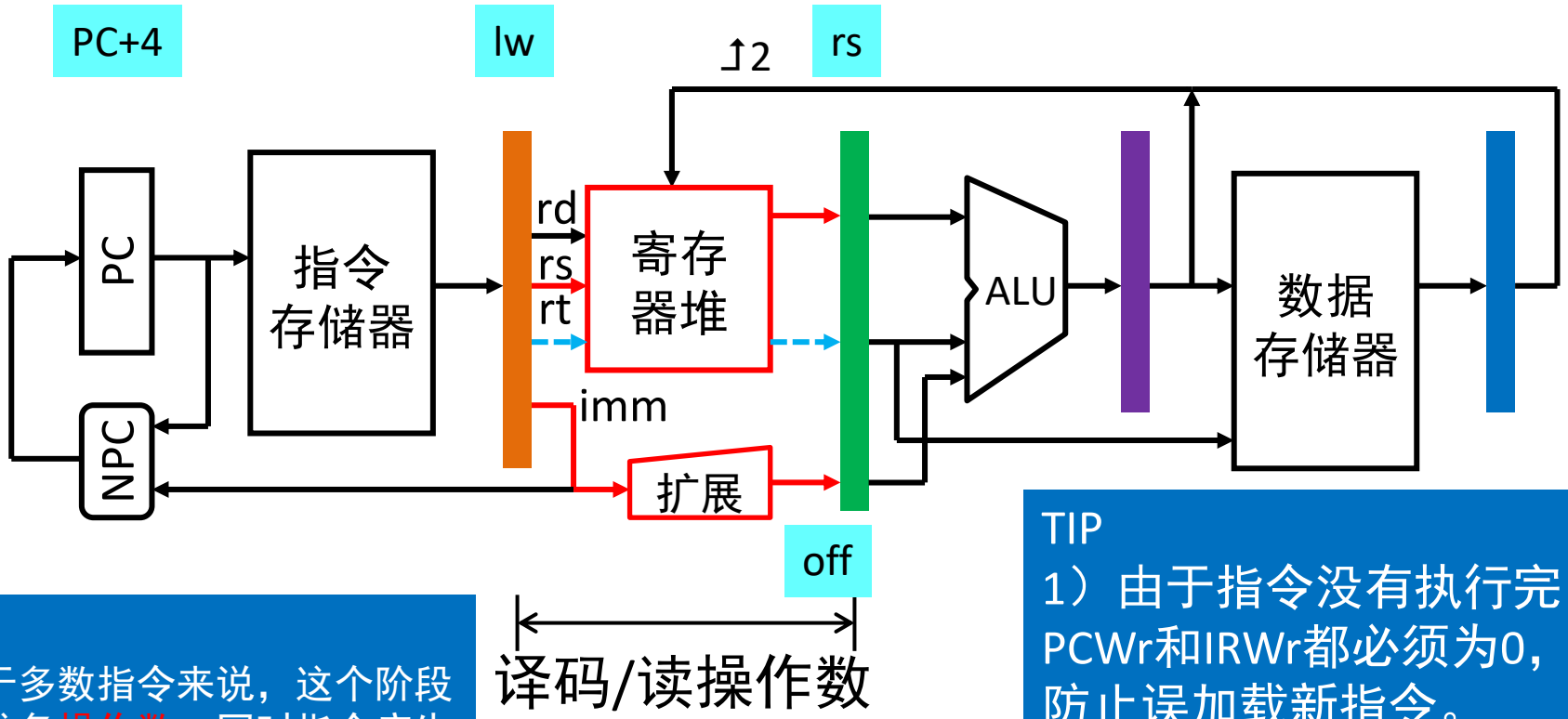
TIP
取指令：所有指令的公共环节
时序：时钟↴后，IR装入lw，PC指向lw下一条指令



R[rt]←MEM[R[rs]+sign_ext(imm16)]

指令执行案例分析：lw

- Cycle 2：读取寄存器&符号扩展
  - IR[25:21]驱动寄存器堆，读出rs并写入A
  - IR[15:00]驱动扩展单元，符号扩展结果写入ER

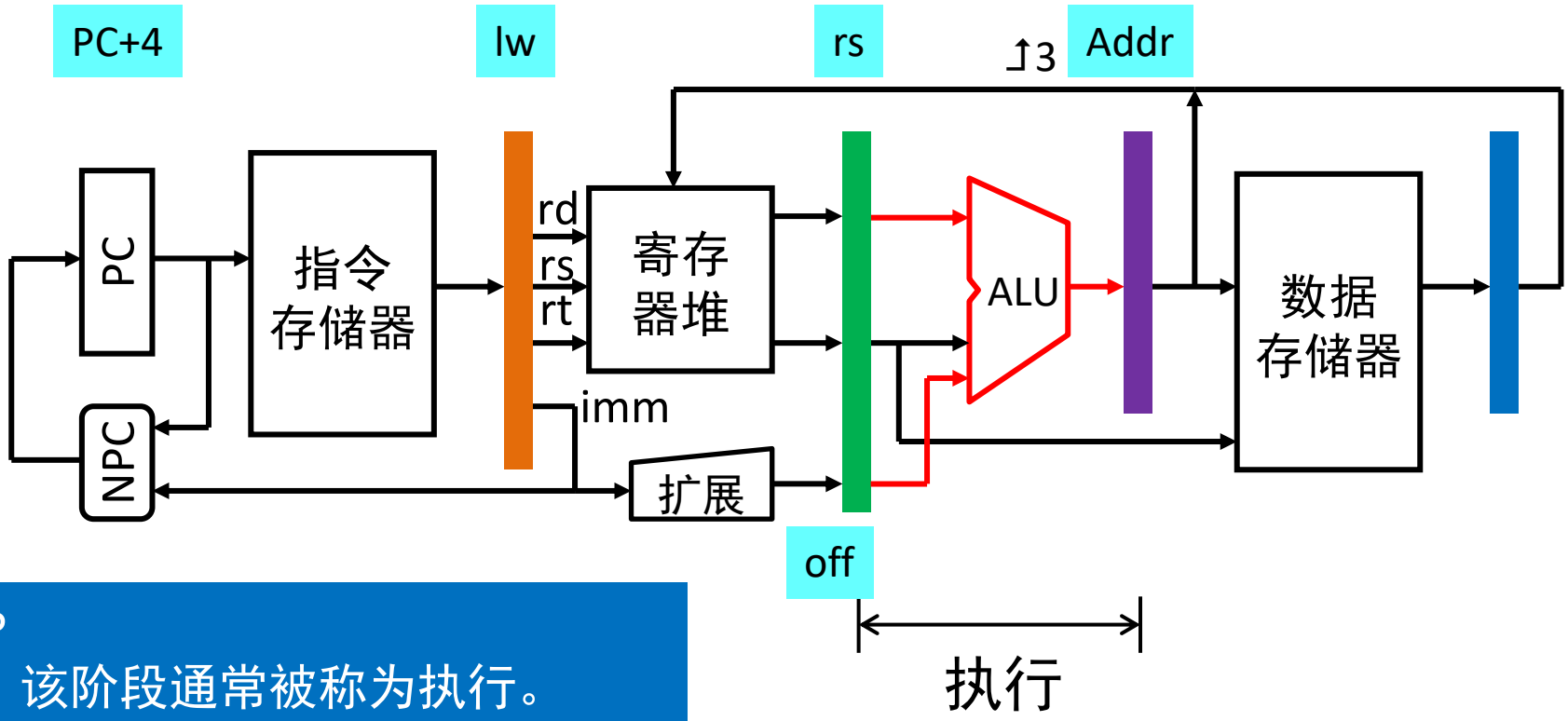


TIP
对于多数指令来说，这个阶段是准备操作数。同时指令产生有效译码结果，因此通常把该阶段称为译码/读操作数

TIP
1）由于指令没有执行完，PCWr和IRWr都必须为0，以防止误加载新指令。
2）寄存器堆仍然会读出rt寄存器的值，只是没有意义。

指令执行案例分析：lw

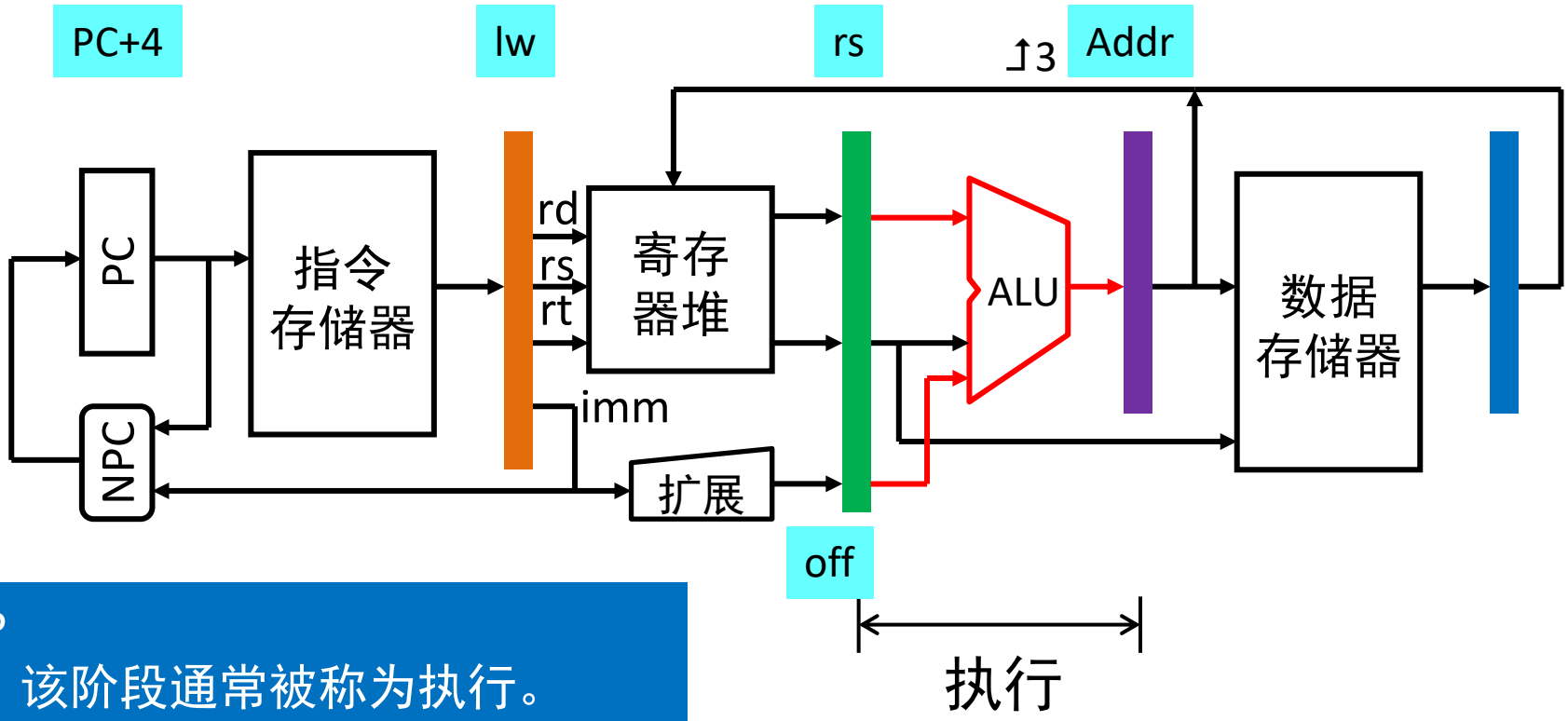
- Cycle 3：ALU计算地址
  - ALU执行加法，计算出的地址写入ALUOut



TIP
1）该阶段通常被称为执行。
2）与前阶段相同，由于指令没有执行完，PCWr和IRWr都必须为0。

指令执行案例分析：lw

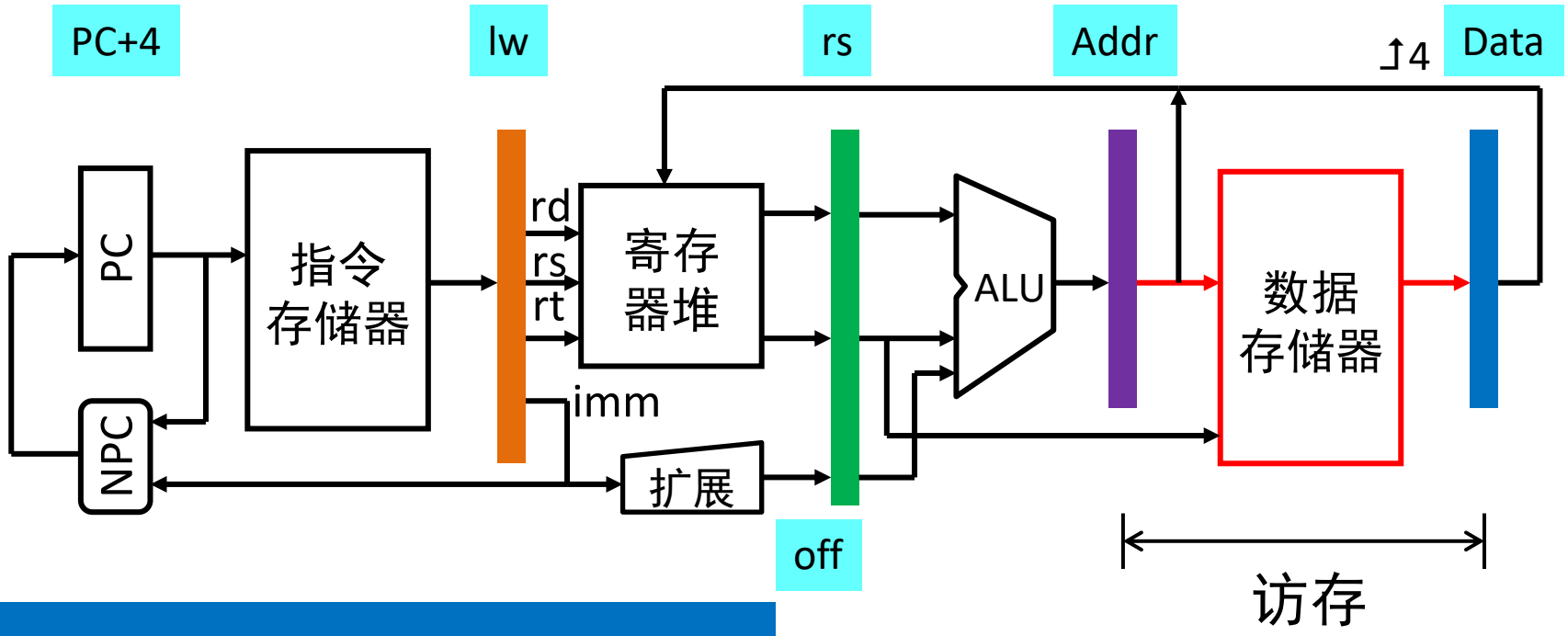
- Cycle 3：ALU计算地址
  - ALU执行加法，计算出的地址写入ALUOut



TIP
1）该阶段通常被称为执行。
2）与前阶段相同，由于指令没有执行完，PCWr和IRWr都必须为0。

指令执行案例分析：lw

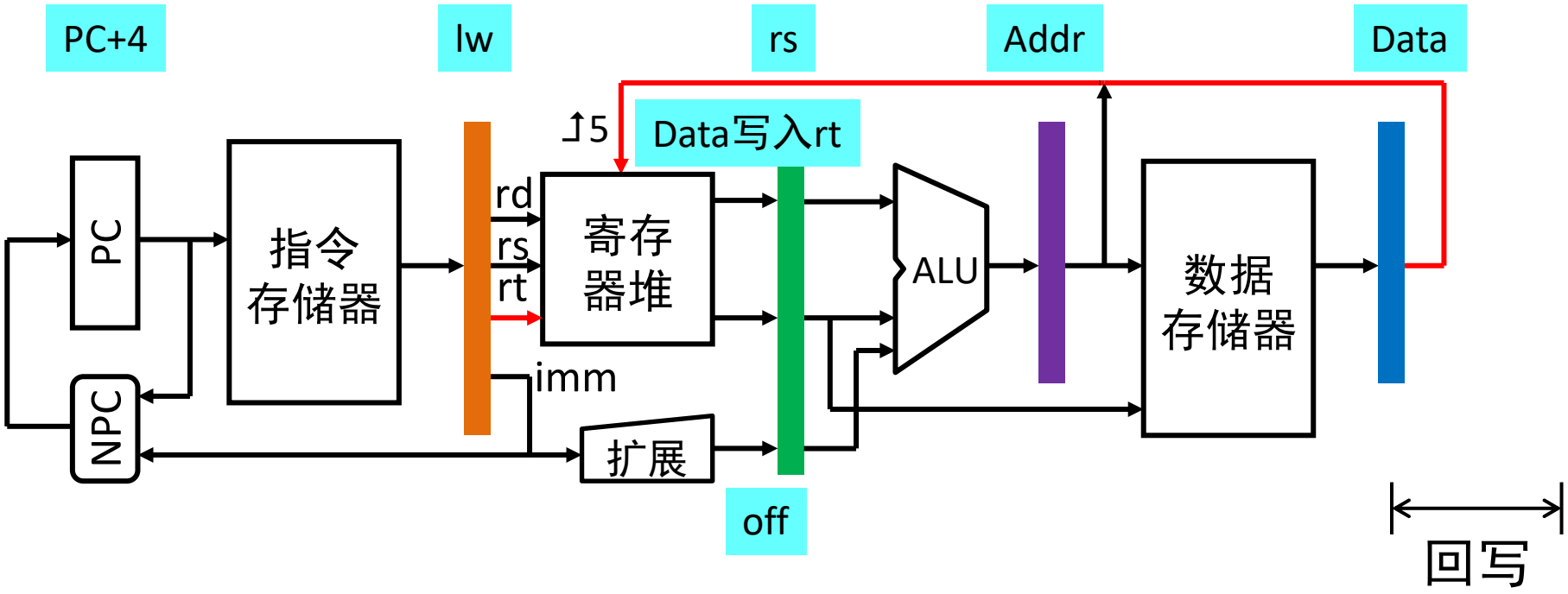
- Cycle 4：读存储器
  - ALUOut驱动DM读出数据，并写入DR



TIP
对于load类指令，该阶段为读存储器；
对于store类资料，该阶段为写存储器。
该阶段通常被称为访存

指令执行案例分析：lw

- Cycle 5：数据写入寄存器堆
  - DR和IR[21:16]驱动RF，数据写入rt



功能部件及其控制信号使用约束

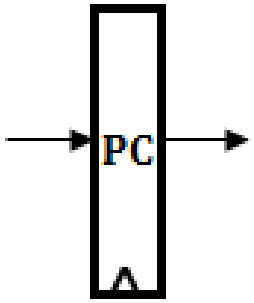
- PC、IR、RF、DM：需要写使能
  - PCWr/IRWr/RFWr/DMWr：只能在特定时间有效！其他时间必须无效！
- A/B/ER、ALUOut、DR：不需要写使能
  - 随时可写

目录

- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

功能部件建模：PC

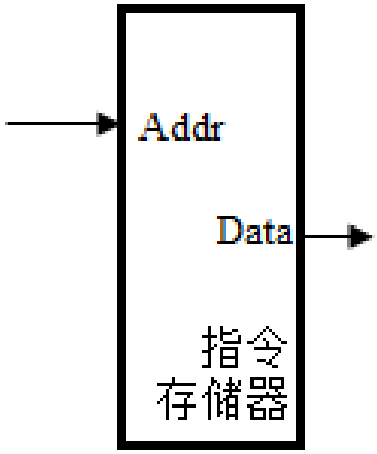
- 功能与控制
  - PCWr：写使能
    - 1条指令执行周期不止1个周期，因此只能允许在特定周期写入NPC值



输入	NPC[32:0] PCWr, Clk, RST	
输出	PC[31:2]	
数据结构	addr，30位寄存器	
行为	功能	操作
	计数输出	PC ← addr
	异步复位	if RST then addr ← 32'h40000
	同步加载	Clk上升沿时： if PCWr then addr ← NPC

功能部件建模：IM

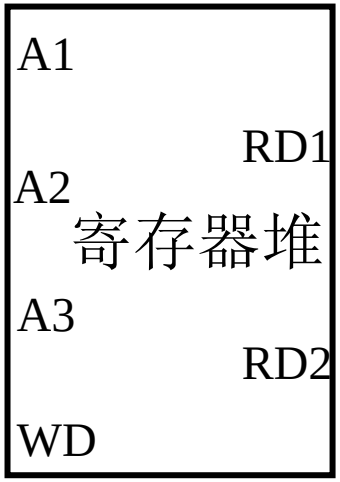
- 数据宽度：32位
- 存储容量：可以任意大
  - 仿真：不超过1K字为宜
- 功能与控制
  - 无需控制，数据经过的固定延迟后输出



输入	AIn[31:0]	
输出	DO[31:0]	
数据结构	Mem为32位×XX字的只读存储器	
行为	功能	操作
	数据输出	DO ← Mem[AIn]

功能部件建模：寄存器堆

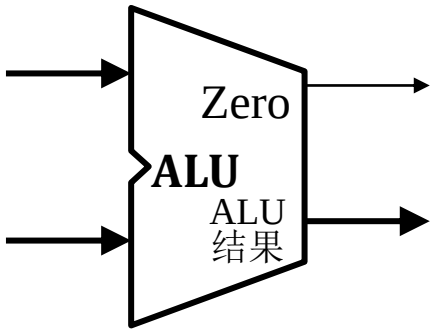
- 功能与控制
  - 读出：不需要控制
  - 写入：RFWr为1



输入	A1[4:0]、A2[4:0]、A3[4:0]、WD[31:0] RFWr、Clk	
输出	RD1[31:0]、RD2[31:0]	
数据结构	RF[0..31]，32个32位寄存器	
行为	功能	操作
	读出寄存器值	RD1←RF[A1]；RD2←RF[A2]
	写入寄存器值	Clk上升沿时 if (RFWr) then RF[A3]←WD

功能部件建模：ALU

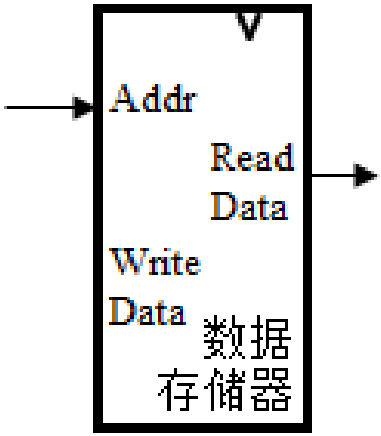
- 功能与控制
  - ALUOp[3:0]决定执行何种计算
- Zero：判断A是否等于B



输入	A[31:0], B[31:0], ALUOp[3:0]		
输出	C[31:0], Zero		
行为	ALUOp	功能	操作
	----	A等于B?	Zero ← (A==B) ? 1 : 0
	0000	加	C ← A + B
	0001	减	C ← A – B
	0010	与	C ← A & B
	0011	或	C ← A | B
	0100	异或	C ← A ^ B
	...	...	...

功能部件建模：DM

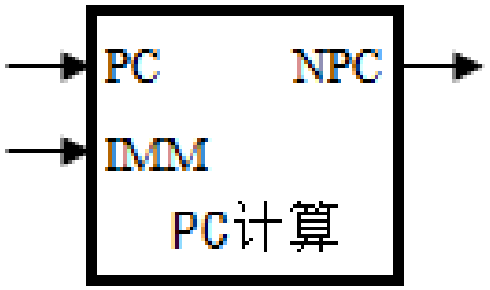
- 数据宽度：32位
- 存储容量：可以任意大
  - 仿真：不超过1K字为宜
- 功能与控制
  - 读出：不需要控制
  - 写入：DMWr为1



输入	AIn[31:0]; WD[31:0] DMWr; Clk	
输出	DO[31:0]	
数据结构	Mem为32位×XX字的存储器	
行为	功能	描述
	读存储	DO ← Mem[AIn]
	写存储	Clk上升沿时 if DMWr then Mem[AIn] ← WD

功能部件建模：NPC

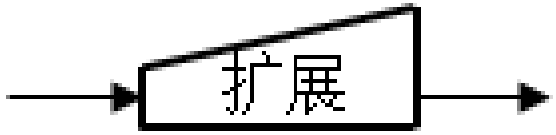
- 计算下条指令的地址
- 功能与控制
  - NPCOp[1:0]决定如何计算PC



输入	PC[31:2], Imm[25:0] NPCOp[1:0]		
输出	NPC[31:2]		
行为	NPCOp	功能	操作
	00	顺序地址	NPC ← PC + 4
	01	计算B指令转移地址	NPC ← PC + sign_ext(imm16) || 00
	10	计算J类指令转移地址	NPC ← PC[31:28] || imm26 || 00
	11	...	...

功能部件建模：EXT

- 立即数扩展
- 功能与控制
  - EXTOp[1:0]决定如何扩展



输入	Imm16[15:0] EXTOp[1:0]		
输出	Imm32[31:0]		
行为	EXTOp	功能	操作
	00	无符号扩展	IMM32 ← {16'b0 || Imm16}
	01	符号扩展	IMM32 ← {{16{Imm16[15]}}, Imm16}
	10	高位扩展	IMM32 ← {Imm16 || 16'b0}
	11		。。。

目录

- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

RTL描述

R(读)→CL(逻辑)→R(写)

- RTL： Register Transfer Language
  - 描述CPU在执行指令时其内部的具体步骤(step)
  - 步骤：包括操作(operation)、寄存器间的通信以及步骤间的时序关系
- 描述方法：
  - 串行描述：step1、step2、step3是顺序执行，每个step执行1个cycle
  - 并行描述：op2、op3是同时执行，并在同一个cycle完成

```
step1: op1
step2: op2; op3
step3: op4
```

多周期数据通路RTL建模的基本思路

- 1、每条指令映射到3～5个cycle
- 2、通过寄存器实现信息传递的相关operation必须部署在不同的cycle
  - 这是由于分段数据通路的依赖性决定的
- 3、可以提早执行的operation就“尽早执行”
  - 门电路被实例化后，就物理的存在，故会始终工作
  - 与其连接的其他电路只是根据条件决定是否采用其输出值而已
  - 因此“尽早执行”而其他电路不采用其值也无所谓

多周期数据通路RTL建模的基本思路

- 4、Cycle1：该周期只能是取指令
  - PC驱动IM读出指令，并将指令写入IR
  - 根据“尽早执行”原则，应同时执行PC←PC+4
- 5、Cycle2：译码信号有效，并同时产生操作数
  - 1）IR存入指令后，控制器就可以产生有效译码结果
  - 2）IR有效，意味着RF必然输出rs值和rt值并写入A/B
    - 当然，A/B存入的可能不是有意义的值（例如对于jal）
  - 3）IR同时也驱动扩展单元，并将扩展立即数写入ER
    - 同理，ER存入的可能不是有意义的值（例如对于add）

指令RTL描述表

- 每行仅刻画1个cycle内的操作：对应R-C-W模型
  - 功能部件：在该cycle需要控制的功能部件
    - 空：代表该cycle不涉及功能部件
  - 控制信号：功能部件的控制信号在该cycle的取值

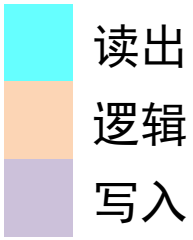
周期	步骤	语义	RTL	功能部件	控制信号

- 下面以addu为例，用倒推法建立addu多周期RTL
  - -1：最后cycle；-2：倒数第2个cycle；其余依次类推
  - 写使能信号：默认值为0
  - 其他控制信号：默认值为X

TIP：
控制信号采用这种表述方法，目的是为了减少表述量

ADDU：RTL建模（倒推1）

- 最后cycle必是ALUOut回写寄存器
- Cycle -1：回写
  - Op1：ALUOut写入rd寄存器



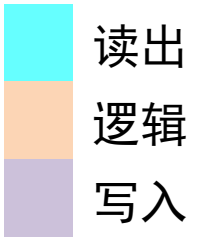
R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	ER				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-1	回写	计算结果回写至rd寄存器	RF[rd]←ALUOut	RF	RFWr←1

R[rd]←R[rs]+R[rt]

ADDU：RTL建模（倒推2）

- 计算结果来自ALU，计算的数据来自A/B
- Cycle -2：计算
  - Op1：A/B驱动ALU计算，结果存入ALUOut



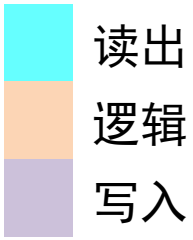
R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	ER				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-2	计算	执行加法，结果存入ALUOut	ALUOut←ALU(A,B)	ALU	ALUOp:ADD

R[rd]←R[rs]+R[rt]

ADDU：RTL建模（倒推3）

- A/B保存的是RF读出的rs和rt
- Cycle -3：读操作数
  - Op1：IR[25:21]驱动RF输出rs，存入A
  - Op2：IR[20:16]驱动RF输出rt，存入B



R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	ER				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-3	读操作数	2个操作数存入A/B	A←RF[rs]; B←RF[rt]	RF	无需控制

R[rd]←R[rs]+R[rt]

ADDU：RTL建模（倒推4）

- 写入IR的一定是指令
- Cycle -4：取指令
  - Op1：读IM，指令写入IR
  - Op2：NPC计算PC+4，并更新PC

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	ER				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-4	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1

R[rd]←R[rs]+R[rt]

LW：RTL建模（倒推）

- Cycle -1：DM回写
  - DR→无功能部件→RF

周期	步骤	语义	RTL	功能部件	控制信号
-1	DM回写	DR写入rt寄存器	RF[rt]←DR	RF	RFWr←1

- Cycle -2：读DM
  - ALUOut→DM→DR

周期	步骤	语义	RTL	功能部件	控制信号
-2	读DM	读取DM，数据存储DR	DR←DM[ALUOut]		

R[rt]←MEM[R[rs]+sign_ext(imm16)]

LW：RTL建模（倒推）

- Cycle -3：计算地址
  - A/ER→ALU→ALUOut

周期	步骤	语义	RTL	功能部件	控制信号
-3	计算地址	执行加法，结果存入ALUOut	ALUOut←ALU(A,ER)	ALU	ALUOp←ADD

- Cycle -4：读操作数
  - IM[25:21]→RF→A；IM[15:0]→RF→ER

周期	步骤	语义	RTL	功能部件	控制信号
-4	DCD/RF 读操作数	基地址存入A； 偏移符号扩展	A←RF[rs] ER←EXT(Imm16)	EXT	EXTOp←SE

R[rt]←MEM[R[rs]+sign_ext(imm16)]

LW：RTL建模（倒推）

- Cycle -5：取指令
  - PC→IM→IR；PC→NPC→PC

周期	步骤	语义	RTL	功能部件	控制信号
-5	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1

R[rt]←MEM[R[rs]+sign_ext(imm16)]

LW：RTL描述表

- 指令时间：5个周期
- 执行路径：→IR→A/Ext→ALUOut→DR→RF

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr←1; NPCOp←+4; PCWr←1
2	读操作数	基地址存入A； 偏移符号扩展	A←RF[rs] ER←EXT(Imm16)	EXT	EXTOp←SE
3	计算地址	执行加法，结果存入ALUOut	ALUOut←ALU(A,EXT)	ALU	ALUOp←ADD
4	读存储器	读取DM，数据存储DR	DR←DM[ALUOut]		
5	回写	DR写入rt寄存器	RF[rt]←DR	RF	RFWr←1

R[rt]←MEM[R[rs]+sign_ext(imm16)]

BEQ：RTL建模（1）

- Cycle 1：取指令
  - Op1：读IM，指令写入IR
  - Op2：NPC计算PC+4，并更新PC

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	ER				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1

```
PC ← (R[rs]==R[rt]) ?
     PC+4+(sign_ext(imm16)||00) :
     PC+4
```

BEQ：RTL建模（2）

- Cycle 2： 读操作数
  - Op1： IR[25:21]驱动RF输出rs，存入A
  - Op2： IR[20:16]驱动RF输出rt，存入B

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	ER				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
2	读操作数	2个操作数存入A/B	A←RF[rs]; B←RF[rt]	RF	无需控制

```
PC ← (R[rs]==R[rt]) ?
     PC+4+(sign_ext(imm16)||00) :
     PC+4
```

BEQ：RTL建模（3）

- 问题：cycle3可否条件写PC？
  - 取决于：1）分支地址是否就绪；2）Zero是否就绪？
- **分支地址**： NPC于cycle2就完成计算
  - cycle1：beq存入IR；PC+4存入PC
  - cycle2：因为NPCOp/imm已有效，所以可计算
- **Zero**：ALU于cycle3产生有效状态

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	ER				DM(写)		

- 结论：PC写入的前提条件在cycle3均已产生

```
PC ← (R[rs]==R[rt]) ?
     PC+4+(sign_ext(imm16)||00) :
     PC+4
```

BEQ：RTL建模（3）

- Cycle 3：条件写PC
  - Op1：ALU比较A和B，根据Zero决定是否写PC

读出
逻辑
写入

R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	**ALU**	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	ER				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
3	条件写分支地址	执行减法，并根据Zero写PC	Zero←ALU(A,B) PC←NPC(PC,imm16)	ALU NPC PC	ALUOp: SUB NPC: BNPC **PCWr: Zero**

```
PC ← (R[rs]==R[rt]) ?
     PC+4+(sign_ext(imm16)||00) :
     PC+4
```

BEQ：RTL描述表

- 指令时间： 3个cycle
- 执行路径：→IR→A/B→PC
  - ①PC+4：cycle1；②PC+4+偏移：cycle3

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr:1 NPCOp:+4 PCWr:1
2	读操作数	RS操作数存入A; 32位无符号扩展	A←RF[rs]; B←RF[rt]		
3	计算并写分支地址	执行减法，判断Zero	ALUOut←ALU(A,B) PC←NPC(PC,imm16)	ALU NPC PC	ALUOp:SUB NPC:BNPC PCWr:Zero

```
PC ← (R[rs]==R[rt]) ?
     PC+4+(sign_ext(imm16)||00) :
     PC+4
```

SW：RTL描述表

- 指令时间：4个周期
- 执行路径：→IR→A/Ext→ALUOut→DM

周期	步骤	语义	RTL	功能部件	控制信号
1	Fetch 取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1
2	DCD/RF 读操作数	基地址存入A； 偏移符号扩展	A←RF[rs] ER←EXT(Imm16)	EXT	EXTOp:SE
3	MA 计算地址	执行加法，结果存入ALUOut	ALUOut←ALU(A,EXT)	ALU	ALUOp:ADD
4	MW 写存储器	rt寄存器写入DM	DM[ALUOut]←RF[rt]	DM	DMWr:1

MEM[R[rs]+sign_ext(imm16)] ← R[rt]

ORI：RTL描述表

R[rt]←R[rs]+zero_ext(imm16)

- 指令时间：4个周期
- 执行路径：→IR→A/EXT→ALUOut→RF

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr:1 NPCOp:+4 PCWr:1
2	读操作数	操作数存入A 无符号扩展	A←RF[rs]; ER←EXT(Imm16)	EXT	EXTOp:UE
3	执行	执行加法，结果存入ALUOut	ALUOut←ALU(A,EXT)	ALU	ALUOp:OR
4	回写	计算结果回写至rt寄存器	RF[rt]←ALUOut	RF	RFWr:1

JAL：RTL建模分析

- 更新PC的前提：
  - PCWr： IR存入jal后就可以产生
  - imm26：IR存入jal后就可以产生
- 回写寄存器的前提：
  - RFWr：IR存入jal后就可以产生
  - PC+4：第1个cycle后，PC+4就计算完毕
- 以上分析表明：写寄存器&更新PC，仅仅依赖于IR是否存储了jal

PC ← PC[31:28] || imm26 || 00
R[31] ← PC+4

JAL：RTL描述表

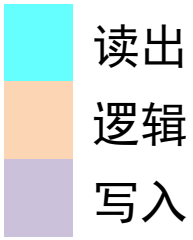
- 指令时间：2个cycle
- 执行路径：→IR→PC/RF
  - jal存入IR后，就可以启动写寄存器和更新PC

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	读取指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr:1 NPCOp:+4 PCWr:1
2	跳转	计算并保存转移PC; 保存PC	RF[31]←PC PC←NPC(PC,imm26)	RF NPC PC	RFWr:1 NPCOp:JNPC PCWr:1

PC ← PC[31:28] || imm26 || 00
R[31] ← PC+4

LUI：RTL建模（倒推1）

- Cycle -1：回写寄存器
  - Op1：ER写入rt寄存器



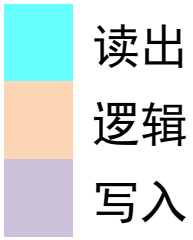
R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	ER				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
1	回写	扩展数据回写至rt寄存器	RF[rt]←ER	RF	RFWr:1

$R[rt] \leftarrow imm16 \,||\, 0^{16}$

LUI：RTL建模（倒推2）

- Cycle -2：高位扩展
  - Op1：IR[15:0]驱动EXT高位扩展，结果写入ER



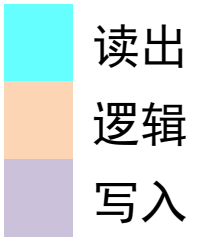
R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	ER				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
1	高位扩展	高位扩展，结果写ER	ER←EXT(Imm16)	EXT	EXTOp:HE

$R[rt] \leftarrow imm16 \,||\, 0^{16}$

LUI：RTL建模（倒推3）

- Cycle -3：读取指令
  - Op1：读IM，指令写入IR
  - Op2：NPC计算PC+4，并更新PC



R	C	R	C	R	C	R	C	R	C	R
PC	IM	IR	RF	A/B	ALU	ALUOut	DM	DR		RF(写)
	NPC	PC(写)	EXT	ER				DM(写)		

周期	步骤	语义	RTL	功能部件	控制信号
-3	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1

$R[rt] \leftarrow imm16 \,||\, 0^{16}$

LUI： RTL描述表

- 指令时间：3个cycle
- 执行路径：→IR→ER→RF
  - jal存入IR后，就可以启动写寄存器和更新PC

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	IR NPC PC	IRWr:1 NPCOp:+4 PCWr:1
2	高位扩展	高位扩展，结果写ER	ER←EXT(Imm16)	EXT	EXTOp:HE
3	回写	扩展数据回写至rt寄存器	RF[rt]←ER	RF	RFWr:1

$R[rt] \leftarrow imm16 \,||\, 0^{16}$

LUI：RTL描述表（4cycle）

- 指令时间：4个cycle
- 执行路径：→IR→A/EXT→ALUOut→RF
- 借用R型计算
  - 注意$0的用途：可以体会MIPS指令格式定义的巧妙

周期	步骤	语义	RTL	功能部件	控制信号
1	取指令	从IM读出指令； 计算下条指令地址	IR←IM[PC]; PC←NPC(PC)	NPC PC IR	IRWr:1 NPCOp:+4 PCWr:1
2	读操作数	$0存入A 无符号扩展	A←RF[0] ER←EXT(Imm16)	EXT	EXTOp:HE
3	执行	执行**OR**，结果存入ALUOut	ALUOut←ALU(A,ER)	ALU	ALUOp:OR
4	回写	计算结果回写至rt寄存器	RF[rt]←ALUOut	RF	RFWr:1

目录

- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

时序分析的目的

- 1、时序是理解难点之一
  - 理解不正确，设计/调试就易于出错
- 2、有利于后续状态机的设计与理解
  - 更好的理解寄存器的数据准备与数据写入的关系
- 3、多周期时序复杂度适中，适于讲解时序
  - 变化：PC、IR、A/B、DR、RF、DM会发生变化
  - 稳定：在1条指令内只变化1次(PC除外)
- 4、进一步训练学生学习形式建模的方法，提高抽象思维能力

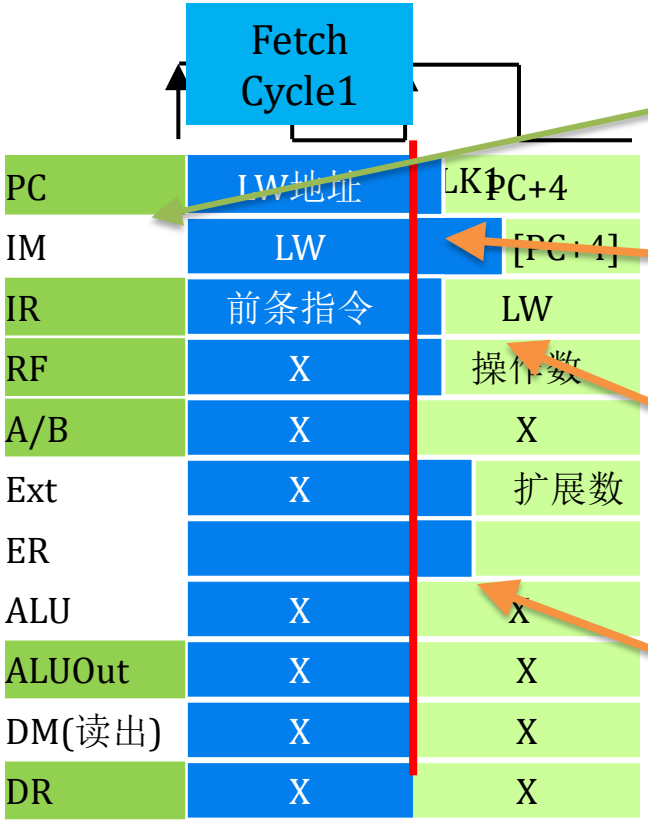
时序分析要点：RTL制导

- RTL制导分析需要哪些必要环节
  - 并非所有环节都需要(多周期的基本特点)
  - 只需关注前序寄存器与后继寄存器间关系
    - 前序值在时钟沿到来后写入寄存器
- ☞注意时钟周期的概念
  - 2个时钟边沿之间的时间

LW时序分析：Cycle1

- Cycle1：取指令(公共周期)
  - 读取IM至IR；同时PC自增4
- X：代表其值无意义
- 边沿后的颜色：代表延迟

周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] ER←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



绿色：功能部件为寄存器

寄存器输出延迟最短

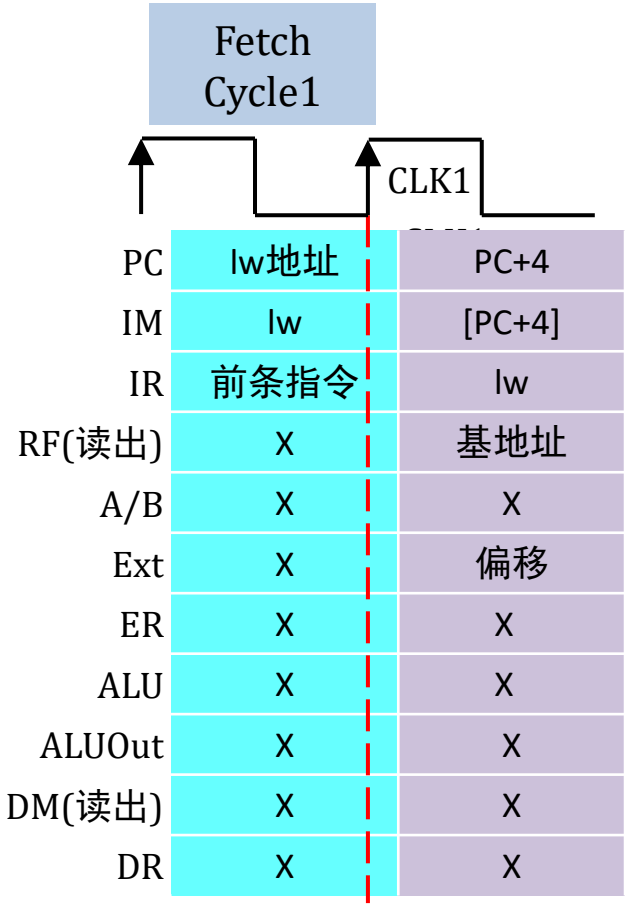
存储器输出延迟比较长

功能部件输出延迟 =
寄存器输出延迟 + 扩展延迟

LW时序分析：Cycle1

- Cycle1：取指令(公共周期)
  - 读取IM至IR；同时PC自增4
- X：代表其值无意义
- 边沿后的颜色：代表延迟

周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] ER←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



TIP

Cycle1之后，IR存入了lw。于是在clk1上升沿之后（即cycle2），在IR驱动控下：

1）控制器开始译码产生各种控制信号

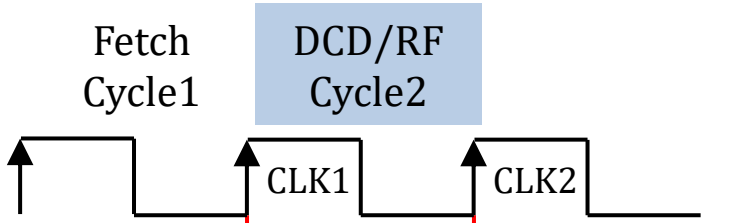
2）RF读出2个寄存器值

3）EXT产生32位扩展数

LW时序分析：Cycle2

- Cycle2：读操作数(公共周期)
  - 读操作数，同时开始译码
  - 扩展单元产生32位扩展数

周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] ER←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR

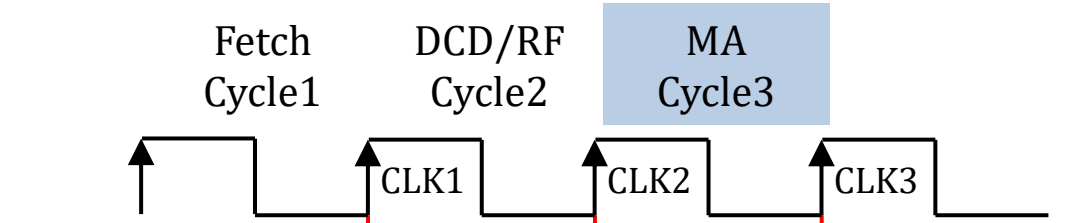


	Fetch Cycle1	DCD/RF Cycle2	
PC	lw地址	PC+4	PC+4
IM	lw	[PC+4]	[PC+4]
IR	前条指令	lw	lw
RF(读出)	X	基地址	基地址
A/B	X	X	基地址
Ext	X	偏移	偏移
ER	X	X	偏移
ALU	X	X	DM地址
ALUOut	X	X	X
DM(读出)	X	X	X
DR	X	X	X

LW时序分析：Cycle3

- Cycle3：计算地址
  - 计算地址并存入ALUOut

周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] ER←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR

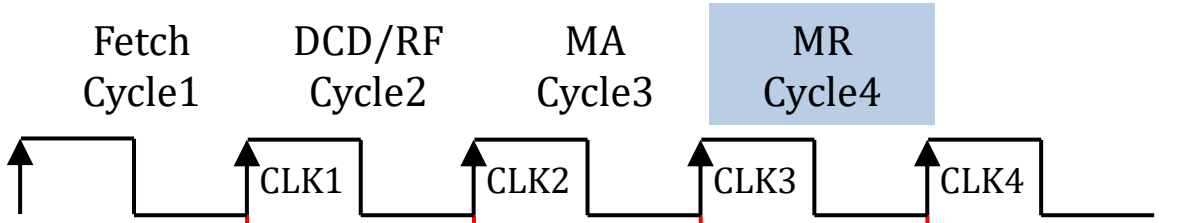


	Fetch Cycle1	DCD/RF Cycle2	MA Cycle3	
PC	lw地址	PC+4	PC+4	PC+4
IM	lw	[PC+4]	[PC+4]	[PC+4]
IR	前条指令	lw	lw	lw
RF(读出)	X	基地址	基地址	基地址
A/B	X	X	基地址	基地址
Ext	X	偏移	偏移	偏移
ER	X	X	偏移	偏移
ALU	X	X	DM地址	DM地址
ALUOut	X	X	X	DM地址
DM(读出)	X	X	X	数据
DR	X	X	X	X

LW时序分析：Cycle4

- Cycle4：读存储器
  - ALUOut驱动DM，数据写入DR

周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] ER←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR

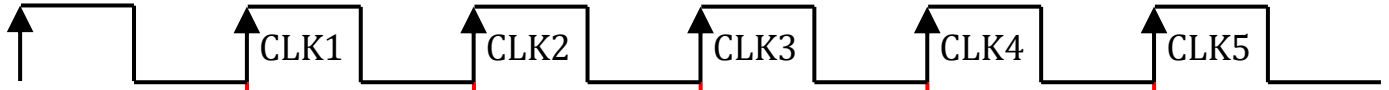


	Fetch Cycle1	DCD/RF Cycle2	MA Cycle3	MR Cycle4	
PC	lw地址	PC+4	PC+4	PC+4	PC+4
IM	lw	[PC+4]	[PC+4]	[PC+4]	[PC+4]
IR	前条指令	lw	lw	lw	lw
RF(读出)	X	基地址	基地址	基地址	基地址
A/B	X	X	基地址	基地址	基地址
Ext	X	偏移	偏移	偏移	偏移
ER	X	X	偏移	偏移	偏移
ALU	X	X	DM地址	DM地址	DM地址
ALUOut	X	X	X	DM地址	DM地址
DM(读出)	X	X	X	数据	数据
DR	X	X	X	X	数据

LW时序分析：Cycle5

- Cycle5：写寄存器
  - DM读出数据写入RF

周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] ER←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR

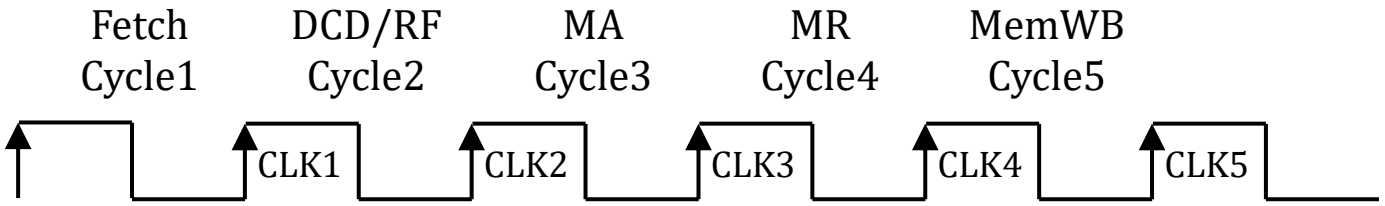


	Fetch Cycle1	DCD/RF Cycle2	MA Cycle3	MR Cycle4	MemWB Cycle5	
PC	lw地址	PC+4	PC+4	PC+4	PC+4	PC+4
IM	lw	[PC+4]	[PC+4]	[PC+4]	[PC+4]	[PC+4]
IR	前条指令	lw	lw	lw	lw	lw
RF(读出)	X	基地址	基地址	基地址	基地址	基地址
A/B	X	X	基地址	基地址	基地址	基地址
Ext	X	偏移	偏移	偏移	偏移	偏移
ER	X	X	偏移	偏移	偏移	偏移
ALU	X	X	DM地址	DM地址	DM地址	DM地址
ALUOut	X	X	X	DM地址	DM地址	DM地址
DM(读出)	X	X	X	数据	数据	数据
DR	X	X	X	X	数据	数据

LW时序分析：简明版

- 可以不考虑延迟
- 只关注前后依赖关系

周期	步骤	RTL
Cycle1	Fetch	IR←IM[PC]; PC←NPC(PC)
Cycle2	DCD/RF	A←RF[rs] ER←EXT(Imm16)
Cycle3	MA	ALUOut←ALU(A,EXT)
Cycle4	MR	DR←DM[ALUOut]
Cycle5	MemWB	RF[rt]←DR



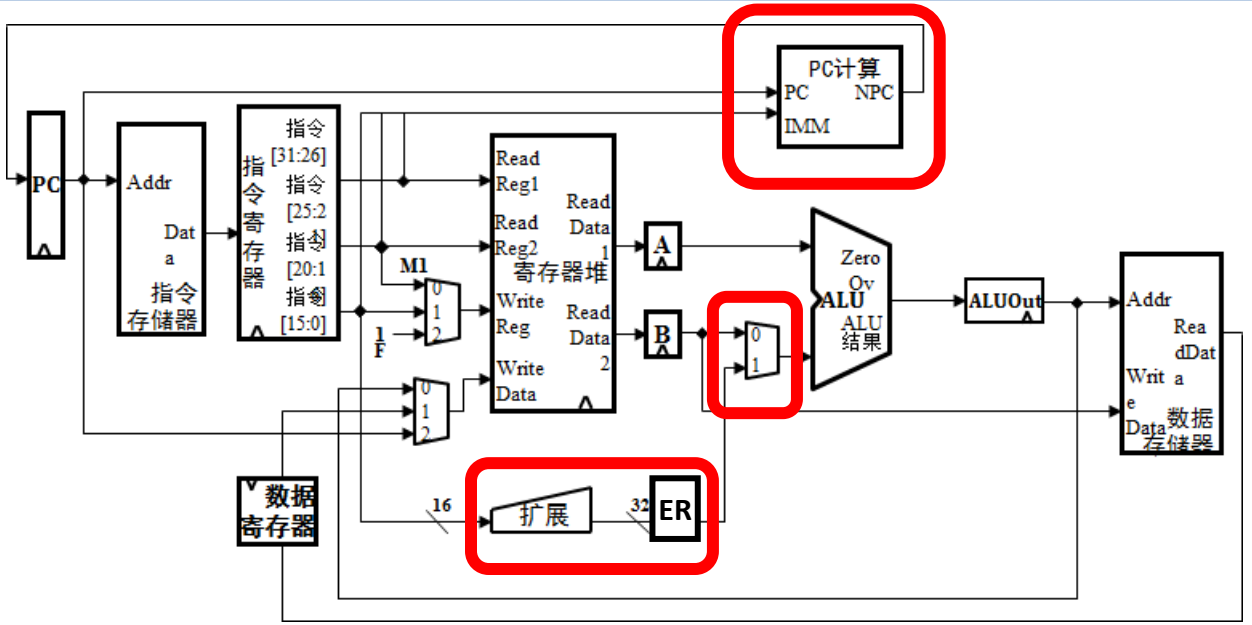
	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle5
PC	lw地址	PC+4	PC+4	PC+4	PC+4	PC+4
IM	lw	[PC+4]	[PC+4]	[PC+4]	[PC+4]	[PC+4]
IR	前条指令	lw	lw	lw	lw	lw
RF(读出)	X	基地址	基地址	基地址	基地址	基地址
A/B	X	X	基地址	基地址	基地址	基地址
Ext	X	偏移	偏移	偏移	偏移	偏移
ER	X	X	偏移	偏移	偏移	偏移
ALU	X	X	DM地址	DM地址	DM地址	DM地址
ALUOut	X	X	X	DM地址	DM地址	DM地址
DM(读出)	X	X	X	数据	数据	数据
DR	X	X	X	X	数据	数据

目录

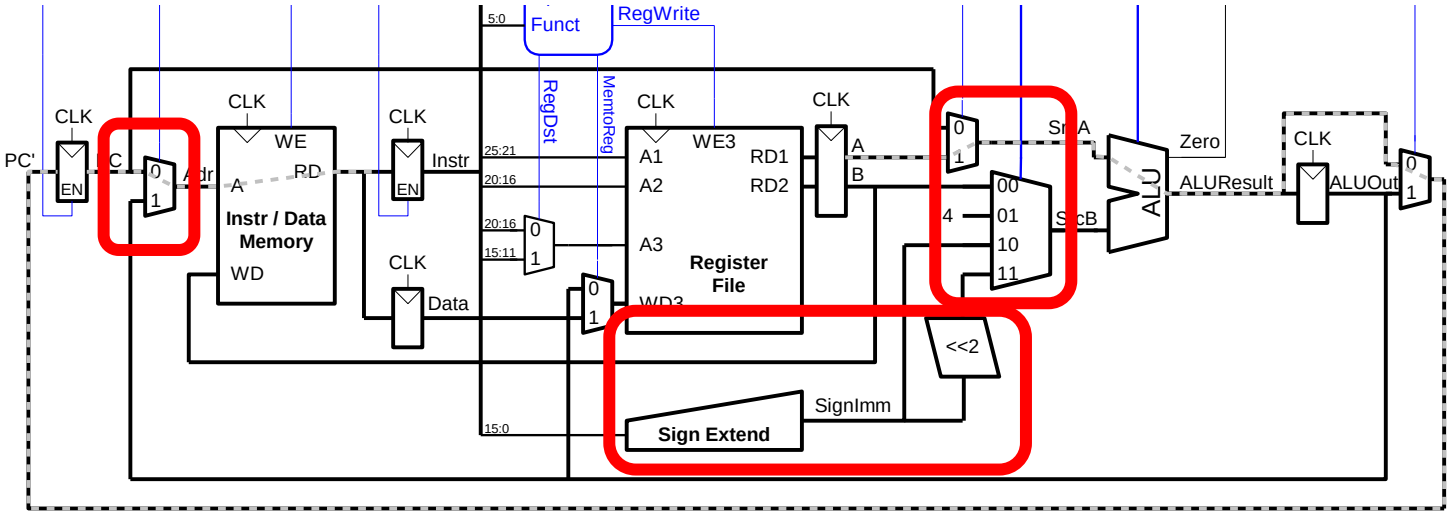
- 从单周期到多周期
- 多周期处理器基本结构
- 功能部件建模
- RTL(Register Transfer Language)
- 基于RTL的时序分析
- 2种数据通路设计对比分析

PPT数据通路 vs 教科书数据通路

PPT
数据通路



教科书
数据通路



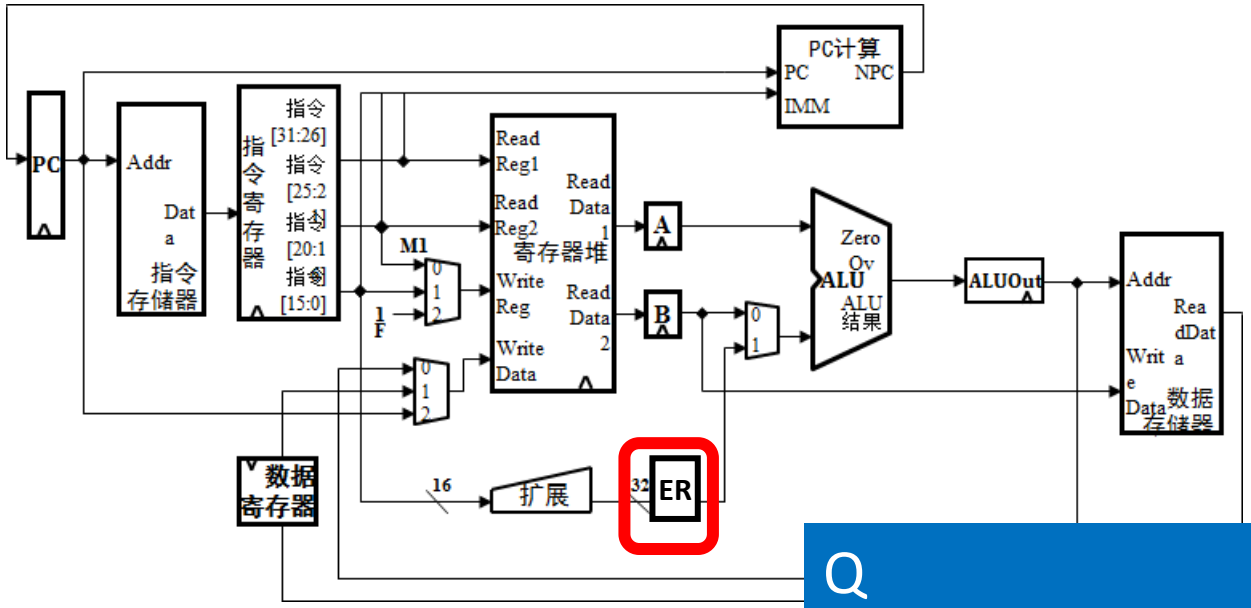
PPT数据通路 vs 教科书数据通路

- PPT：引入NPC，数据通路的结构化特征更好
  - 整个设计更加简洁，体现高内聚低耦合的设计思想
  - 所有与地址产生的功能都放在NPC中（包括今后的c处理器上电启动地址和异常地址）
  - IM地址端，ALU的A端：没有MUX
- 教科书：可节省一点逻辑，但清晰度不好
  - 只有1个ALU，节省了NPC里的加法器
  - 状态机设计更加复杂，更有利于学习“状态机”

PPT数据通路 vs 教科书数据通路

PPT

数据通路



Q

ER的重要性在哪里？

教科书

数据通路